REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE December 1993	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE Exploring Systematic Reuse for Command and Control Systems	5. FUNDING NUMBERS G MDA972-92-J-1018
6. AUTHOR(S) J. O'Connor, C. Mansour, J. Turner-Harris, G. Campbell, Jr. Produced by Software Productivity Consortium under contract to Virginia Center of Excellence.	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Virginia Center of Excellence SPC Building 2214 Rock Hill Road Herndon, VA 22070	8. PERFORMING ORGANIZATION REPORT NUMBER SPC-92020-CMC, Version 02.00.02
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) ARPA/SISTO Suite 400 801 N. Randolph Street Arlington, VA 22203	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Supersedes "Introducing Systematic Reuse to the Command and Control System Division of Rockwell International" DTIC (ADA 252271)

12a. DISTRIBUTION / AVAILABILITY STATEMENT No Restrictions	12b. DISTRIBUTION CODE 1

19961022 069

13. ABSTRACT *(Maximum 200 words)*

This paper describes an initial use and evaluation of the Software Productivity consortium's Synthesis methodology by the command and Control Systems Division (CCSD) of Rockwell International. Synthesis supports instituting a product line of similar software-intensive systems, forming a domain of mechanically derivable systems. The potential for a product line exists whenever there is a perceived market for a series of similar products or product versions. Synthesis defines an approach, based on domain-specific reuse, by which an organization can standardize its perceptions of customers' needs and effective solutions to those needs. The approach helps an organization identify specific business objectives that enable standardized products. Product standardization is the foundation for a product line and an associated production process with which the organization can achieve significant improvements in productivity, product quality, manageability, and responsiveness to diverse and changing customer needs.

14. SUBJECT TERMS Reuse, process, domain, product line, family	15. NUMBER OF PAGES 15
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

EXPLORING SYSTEMATIC REUSE FOR COMMAND AND CONTROL SYSTEMS

SPC-92020-CMC

VERSION 02.00.02

APRIL 1994

EXPLORING SYSTEMATIC REUSE FOR COMMAND AND CONTROL SYSTEMS

SPC-92020-CMC

VERSION 02.00.02

APRIL 1994

James O'Connor, Software Productivity Consortium
Catharine Mansour and Jerri Turner-Harris,
Rockwell International
Grady H. Campbell, Jr.,
Software Productivity Consortium

Produced by the
SOFTWARE PRODUCTIVITY CONSORTIUM SERVICES CORPORATION
under contract to the
VIRGINIA CENTER OF EXCELLENCE
FOR SOFTWARE REUSE AND TECHNOLOGY TRANSFER

SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

Note:

This paper was formatted in accordance with standards for *IEEE Software* submissions.

Introduction

This paper describes an initial use and evaluation of the Software Productivity Consortium's Synthesis methodology[1] by the Command and Control Systems Division (CCSD) of Rockwell International. Synthesis supports instituting a product line of similar software-intensive systems, forming a domain of mechanically derivable systems. The potential for a product line exists whenever there is a perceived market for a series of similar products or product versions. Synthesis defines an approach, based on domain-specific reuse, by which an organization can standardize its perceptions of customers' needs and effective solutions to those needs. The approach helps an organization identify specific business objectives that enable standardized products. Product standardization is the foundation for a product line and an associated production process with which the organization can achieve significant improvements in productivity, product quality, manageability, and responsiveness to diverse and changing customer needs.

In December 1990, CCSD initiated a Consortium-assisted pilot project to evaluate the applicability of the Synthesis methodology to Rockwell's needs. The pilot project had several objectives. For CCSD, it was initially a vehicle for evaluating its needs and opportunities for reuse and the applicability of the Synthesis methodology to those needs. Later, it became the vehicle for transferring Synthesis practices into CCSD and transitioning them into production use. For the Consortium, pilot projects are a primary means for validating the Synthesis approach, a mechanism for disseminating it to adopters, and a source of experience for improving the approach. This pilot project was a formal commitment between CCSD and the Consortium under which the Consortium provided training and consulting to CCSD managers and engineers; CCSD, in turn, assigned experienced managers and engineers who had knowledge and expertise in the targeted CCSD business area and the ability to create quality products representative of that business.

The result of the pilot experience was a decision by CCSD management to attempt to transition Synthesis practices into use in support of production projects. The pilot effort showed that Synthesis was a sound framework for methods and greater discipline in CCSD software development and would give Rockwell competitive advantages in targeted business areas. As a result of this and other pilot projects in industry and government, the Consortium has been able to significantly extend and refine its guidance to organizations attempting to adopt a Synthesis approach.

In this article, we present the CCSD experience in using the Synthesis methodology so far. As a foundation, we first describe the characteristics of a Synthesis process. CCSD application of a Synthesis process has resulted in the production of a partially automated environment that supports the specification of a system in the targeted CCSD domain and the generation of corresponding software requirements, design, and code. A domain-specific notation created by the project lets an engineer describe one of these systems in terms of high-level requirements and engineering decisions. A corresponding product is generated by mechanically selecting, adapting, and composing reusable components based on the decisions expressed in the specification. The utility of this environment has been demonstrated through successful creation of parts of two products and limited use on a current CCSD project.

The Synthesis Process

Synthesis is a methodology for constructing software systems as instances of a family of systems having similar descriptions. The Synthesis approach is an elaboration of the concept of program families[2,3]. Synthesis enables an organization to leverage the commonality among similar systems by developing standardized requirements, design, and corresponding reusable components. Individual projects then use these assets repeatedly in a standardized application development process to rapidly construct a tailored product (i.e., systems and associated deliverable and supporting work products).

This section is an informal description of the Synthesis methodology. The Consortium's *Reuse-Driven Software Processes Guidebook*[4] contains a complete description of Synthesis and detailed guidance on performing a Synthesis process.

A key aspect of Synthesis is that it defines a systematic approach to identifying the commonalities and variabilities that characterize a product line, a potential set of similar products. The commonalities reflect work that can be done once (in the form of reusable components) and reused to create any product within the domain's scope. The variabilities indicate how a product line based on reusable components must be designed so that a tailored product can be mechanically derived, via compile-time parameterization, to satisfy the needs of a customer for a particular system. Based on variabilities, you can define a specialized application development process in which work products (e.g., requirements, design, code, test support, user documentation) are produced simply by the mechanical adaptation and composition of reusable components.

To separate the issues of product standardization, reusable component creation, and continual improvement of the production process from the issues of delivering products to customers, a Synthesis process organizes work into two interacting activities: domain engineering and application engineering (see Figure 1). Domain engineering is an ongoing activity for defining, implementing, and evolving a domain in the form of a product line and a standardized development process. Application engineering is a recurring activity in which projects iteratively perform the standardized process using the product line to produce application products that satisfy their customers' needs. An application engineering project is initiated for each new system (or system version or variant) to be developed in the business area (i.e., whenever a commitment is made to work with a customer, such as at receipt of a request for proposal or upon award of a contract). There may be many of these projects active at one time, all using the same domain engineering product. The application engineering process begins when a customer need for a product is established and continues, through repeated deliveries of the product, until the customer no longer needs that product as originally conceived. Domain engineering continues as long as any such projects are considered viable. Experience gained in application engineering is fed back to domain engineering, where it can be used to improve the supported product line and associated application engineering process.

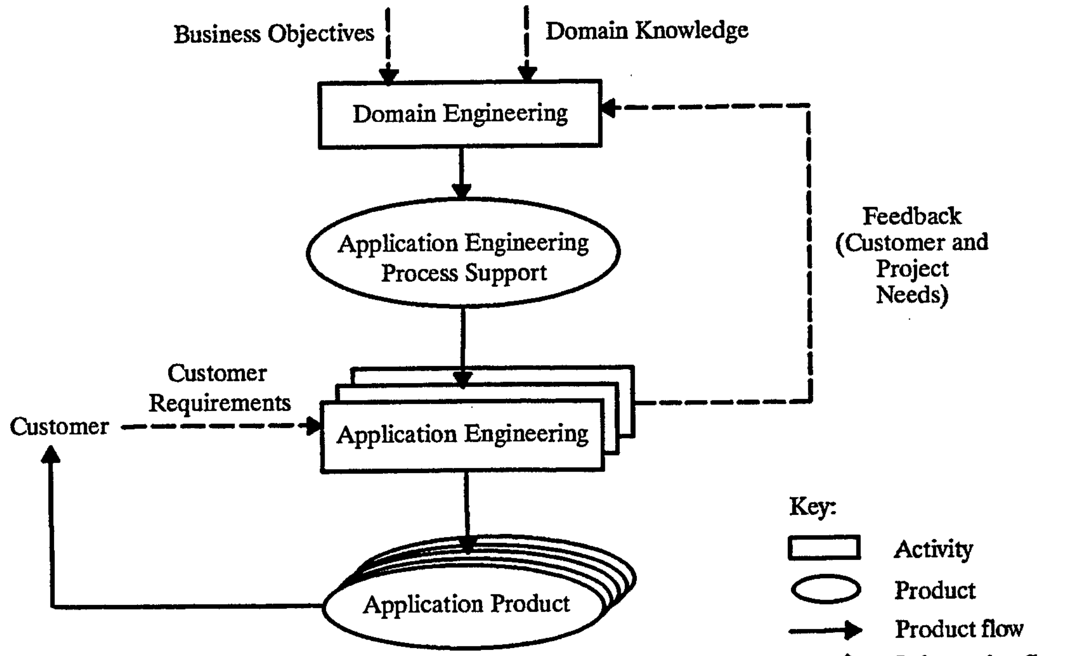


Figure 1. The Synthesis Process

Domain Engineering. Domain engineering is performed to create a product line and support for an associated application engineering process. The activities of domain engineering are domain management, domain analysis, domain implementation, and project support.

Domain management comprises the planning, monitoring, and control of the domain engineering effort. Planning results in a master plan for the development and evolution of the domain and in increment plans for each of a series of domain engineering iterations. Domain management is also concerned with all facets of process management for the domain, including configuration management and quality assurance disciplines.

In domain analysis, you define a domain in a domain definition and formally describe it in a domain specification. Specifically, you perform the following tasks:

- *Define the scope and boundaries of the domain and determine whether the domain is viable.* The domain definition describes what type of systems you want to produce in the future and their general characteristics. The viability analysis determines whether your investment in domain engineering is justified. Factors affecting viability include the degree to which the domain is understood, the amount of commonality (in requirements) between systems in the domain, the volatility of the requirements in such systems, and the size of the market for such systems.

- *Define the critical requirements variations that distinguish systems in the domain.* These variations, formalized in a decision model, provide the basis for a domain-specific language that engineers use to specify systems in the domain. For example, if your domain were office telephone systems, the variations might include the maximum number of extensions and the availability of features such as call waiting, conference call support, and voice mail.

- *Develop standardized requirements for products in the domain.* This is a specification of the requirements for all systems in the domain. It specifies both how requirements are common to all systems in the domain and how they vary from system to system.

- *Develop a standardized design for products in the domain.* A design consists of a standardized (adaptable) architecture, a set of adaptable components, and a mapping that prescribes how the architecture and components are used to generate a tailored product given a particular set of requirements and engineering decisions allowed by the decision model.

- *Describe a standardized application engineering process.* The described process tells the engineer how to specify systems in terms of the decision model and how to use the specification to guide the construction of corresponding deliverables from adaptable components. Because this process is driven by the decision model, it focuses the engineer on the critical requirements variations that differentiate systems in the domain.

- *Verify the work products of domain engineering.* The work products of domain engineering (the domain definition, domain specification, and domain implementation) must be verified as mutually consistent. Each must be self-consistent, the domain specification must conform to the domain definition, and the domain implementation must conform to the domain specification.

The first task above produces a domain definition. The results of the next four tasks together form a domain specification, which elaborates the characteristics of the domain as set by the domain definition.

In domain implementation, you implement support for the products in the domain and the associated production process in conformance with the domain specification. Specifically, you:

- *Implement adaptable components.* Techniques for implementing adaptable components include programming language generics, e.g., Ada generic packages and C++ class templates; word processing merge and macro facilities, e.g., those in WordPerfect or in Microsoft Word; and metaprogramming tools, e.g., Metatool[5] and the TRF2 metaprogramming tool[6].

- *Document the application engineering process.* This creates standard policies and procedures that application engineering projects will follow to attain best practices within a standardized application engineering process as defined in the domain specification. These policies and procedures are tailored to the needs of the organization and its business area.

- *Develop automated support for application engineering (optional).* This provides automation supporting performance of the standardized application engineering process, including project management and specification, evaluation, and generation of products, consistent with policies and procedures documented for that process.

Project support is domain engineering effort associated with assisting application engineering projects in making effective use of the domain. An element of this support is to validate that the domain is responsive to the needs of the projects and their customers. Feedback from projects is a major stimulus for subsequent improvement and evolution of the domain as a viable resource for current and future projects.

Application Engineering. The domain implementation is delivered to application engineering projects as process support. It gives application engineers the ability to specify a system and produce a corresponding product. The engineer creates an application model, which is a specification of the desired system, by resolving the variations (e.g., choosing among alternatives) accommodated by the decision model. For example, in a business telephone systems domain, an engineer might specify a particular telephone system by requesting the basic system (common to all such systems) enhanced to allow a maximum of 50 extensions, call waiting, and voice mail (some of the variations that such a system might support). A different system might support conference calling but not voice mail; another might support both.

The application engineer then uses the application model to direct the selection, adaptation, and composition of adaptable components to construct deliverable or supporting work products. The construction process may be automated or manual; either way, work products are derived mechanically from the specification. By mechanically, we mean that there are precise instructions that describe every detail of how to use the information in an application model to construct work products from adaptable components. Formulating a mechanical construction process is essential to effective automation. In cases in which variations (requirements) needed by a project were not anticipated in the domain analysis or were intentionally not supported in the domain implementation, specialized components may be designed, implemented, and integrated into the final product using conventional techniques. Note that the application engineer never "searches" a reuse library or manually modifies components. The development process indicates exactly which adaptable components are needed and how they are to be adapted to satisfy a particular specification.

An organization realizes productivity gains from a Synthesis approach because systems in the domain can be rapidly specified and constructed using adaptable components. Effective reusability is due to the flexibility of adaptable components and because the process is designed to take best advantage of these components (i.e., the components and the process for their use are concurrently engineered). Product quality improves as a result of following a standardized process in which tailored products are constituted from standardized reusable components.

Pilot Project Experience

The first task of the pilot project was to select a business area in which to practice Synthesis. We chose the area of Communications Control and Management (CCM) systems because of CCSD's experience and interest in this area. Systems in that area were known to have much in common but also to vary in interesting ways. CCM systems perform communication control and station management functions for ground and transportable high frequency (HF) radio stations. These stations contain communication devices (e.g., HF radios, antennas, encryption/decryption devices, modems); electronic switching equipment; and functions that support the configuration, monitoring, and control of station resources. Stations provide communication capabilities to commercial or military subscribers.

Given the breadth of the CCM domain and the limited resources available, application of Synthesis to the entire domain was not feasible. The preferred fall-back strategy is to focus on a subdomain (e.g., the domain corresponding to a CCM subsystem) that can be investigated in sufficient detail. However, selecting the proper subdomain requires some agreement on the nature, scope, and structure of the entire domain. Our goal was to select a subdomain that would both be useful in its own right and would remain useful if expansion to the full domain was later judged to be feasible and desirable. Our approach was to partition the pilot project into four phases:

1. A top-level domain analysis of the entire CCM domain

2. The selection of a subdomain based on the results of the top-level domain analysis

3. A detailed domain analysis of the selected subdomain

4. A partial implementation of the selected subdomain

This approach permitted us to perform a detailed application of Synthesis in a subdomain while, at the same time, giving us confidence that the results could be compatible with future developments of the domain as a whole. To validate the results of this effort, we then applied the domain product to create corresponding parts of the documentation for two CCM systems. Subsequently, we used a part of the domain product in the development of a new system.

The remainder of this section describes our experience in performing the pilot project. This description idealizes our experience to some degree. Synthesis is a strongly iterative process; the details of actually having performed such a process can be confusing to describe accurately. Despite significant additional iteration among and within activities, a more traditional results-oriented view is sufficient to understand the essentials of what was accomplished.

Top-Level Domain Analysis. Our goal for the first phase of the pilot was to define the CCM domain to a sufficient level of detail so that the results could guide the selection of a subdomain as a focus for the second phase.

Domain Definition. The first step in this phase was to produce the domain definition. The major inputs to the domain definition task were our knowledge of the systems Rockwell has produced in the past and our best understanding of what kinds of systems Rockwell customers will want in the future. Decisions to broaden or narrow the domain depended on business factors as much as technical factors. We produced the following work products during this activity:

- *A short (two-page) narrative description of the domain.* This described the domain in terms of the problems that systems in the domain solve, the context in which these systems operate, the basic functions the systems perform, and the systems' users. This was the first product we developed and the one we went to first when we needed to decide if a given system was in the domain or not. It also served as an introduction to the domain for engineers new to the project.

- *A domain glossary.* This defined the terminology that we use to discuss requirements and systems in the domain. This glossary borrowed heavily from a standard glossary of telecommunications terms. We found the domain glossary to be essential for effective communication between the engineers working on the pilot. Having standard terminology removed confusion that occurred early on when different engineers used the same term to refer to subtly different or sometimes very different concepts or entities.

- *A set of commonality and variability assumptions.* These assumptions stated explicitly but informally what is common to systems in the domain and what is variable. An example of a commonality assumption is "all systems in the domain perform equipment diagnostics." Examples of variability assumptions are "the types of diagnostics performed by systems in the domain will vary" and "the initiating event of a diagnostic will vary across systems in the domain (e.g., on power-up, periodic, operator initiated)."

The completed domain definition represented an agreement between the pilot participants on the class of systems included in the domain and the general characteristics of these systems. Drawing on CCSD's experience in marketing and developing similar systems, we concluded that the domain as described in the domain definition was both technically and economically viable.

Domain Specification. The next step in this phase was to develop the domain specification. The primary inputs to this activity were the domain definition and our understanding about how systems in the domain are developed (i.e., specified, designed, and implemented). To achieve our goal of subdomain selection, it was sufficient for us to create the decision model, standardized requirements, and standardized design portions of the domain specification:

- *Decision Model.* We represented our decision model as a formalized set of questions that the engineer is to answer (e.g., "Should power-up diagnostics be performed? If yes, which of the following power-up diagnostics should be performed? [list of choices]"). The decisions on this list were created by elaborating from the variability assumptions in the domain definition.

- *Standardized Requirements.* Our standardized requirements consisted of requirements that are common to all systems, derived from the commonality assumptions, and the requirements that vary from system to system, derived from the variability assumptions. We integrated the common and variable requirements into a single representation by creating a requirements document that was parameterized by the decision model. We parameterized the requirements by annotating them with notations indicating how they should be tailored based on decision model choices. For example, we used the notation to represent conditions such as if power-up diagnostics are selected, include this section, and for each power-up diagnostic type selected, include the appropriate subsection.

- *Standardized Design.* We represented the standardized design as a parameterized module structure[7]. Like the standardized requirements, the standardized design is parameterized by decisions from the decision model. The parameterization allowed modules to be included or excluded from the structure based on decision model choices. The modules in the module structure correspond to adaptable code components (i.e., families of modules).

Subdomain Selection. We used the result of the top-level domain analysis to guide the selection of a subdomain for further investigation. Having a standardized description of the high-level requirements and design for the CCM domain made this a straightforward task. The subdomain we selected was the software that supports communication over the MIL-STD-1553B system bus[8]. We considered a number of other subdomains (e.g., system diagnostics, operator interface), but this subdomain was selected for the following reasons:

- It represented an integral part of systems in the domain.
- It exhibited nontrivial variability.
- It was small enough that it could be investigated in detail.
- The engineers assigned to the pilot project were familiar with the operation of the software.
- The resulting work products would be useful to existing projects.

Subdomain Analysis. The goal for this phase of the pilot project was to define the subdomain in sufficient detail to guide the implementation of reusable components and the creation of a standardized development process for the production of deliverables using those components.

Domain Definition. The domain definition for the subdomain started with the work products of the informal domain definition for the CCM domain (i.e., narrative description, glossary, and assumptions). These work products were developed by extracting the relevant parts from the CCM domain definition and elaborating on them to reflect the more detailed analysis done for the subdomain. As a result of this narrowed definition, we concluded that the subdomain was both technically and economically viable.

Domain Specification. The next step in analyzing the subdomain was to develop the domain specification. Our domain specification consisted of the following work products:

- *Decision Model.* We created the subdomain decision model by extracting the relevant portions from the CCM decision model and elaborating where necessary. Supported variations ranged from high-level decisions regarding the architecture of the target system (e.g., the number and characteristics of the buses, terminals, and subsystems) to very detailed decisions about particular components (e.g., the command word format of a particular bus). The decision model also supported variations that had nothing to do with the operational software, but they were necessary to generate deliverable documentation (e.g., contracting agency and contract number). Decisions in the decision model were structured into groups of related decisions referred to as decision classes. A top-level decision class from the subdomain decision model is shown in Figure 2.

Decision Class:	1553B Subsystem	
Decision Constraint:	0 to 10 per application system	
Decisions	*Value Space*	*Description*
Subsystem Identifier[SSID]	identifier	Unique identifier for a 1553B subsystem
RAM[SSRAM]	hex [(0 .. ~)]	Amount of random access memory
ROM[SSROM]	hex [(0 .. ~)]	Amount of read only memory
Subsystem terminals [STERM1 – STERM10]	list of TERMINALS	Terminals for this subsystem
Processor[SSPROC]	enum of (Intel80186, Intel80286, Intel80386)	Type of subsystem processor

Figure 2. A Decision Class From the Subdomain Decision Model

- *Standardized Requirements.* We represented our standardized requirements in the form of a parameterized Real-Time Structured Analysis (RTSA) specification. This choice was made because we wanted to use the Ada-based Design Approach for Real-Time Systems (ADARTS℠)[9] as a design method, and, at that time, ADARTS required an RTSA specification as input. Variations in the RTSA were represented by having RTSA data transformations (i.e., "bubbles") decompose into multiple versions corresponding to decision model choices.

- *Standardized Design.* We represented the standardized design for the domain as a parameterized ADARTS design. Variation was shown at the architectural level by having multiple diagrams and having portions of diagrams decompose into multiple versions depending on decision model choices. This technique, although limited, worked because the architecture was relatively invariant with respect to decision model choices. It is worth noting that representing variation in graphical design methods is a very difficult problem. Variation was represented in the specification of adaptable components using parameterized program definition language (PDL).

• *Standardized Development Process.* We defined an automated process for specifying application systems and for generating deliverables from the specifications. We chose an automated process rather than a manual one because our experience was that automated processes are more readily accepted by the targeted users (i.e., in-house engineers) and would make them more receptive to the concepts of Synthesis.

Subdomain Implementation. The goal for this phase of the pilot project was to partially implement the subdomain so that selected deliverables could be constructed from reusable components. Our subdomain implementation consisted of the following products:

• *Adaptable Requirements Components.* These components are used to produce software requirements specifications for systems in the subdomain. The standardized requirements developed during the subdomain analysis determined the content of these components. Customer requirements determined the form of the work products. We used WordPerfect to create requirements components that could be automatically adapted based on decision model choices. WordPerfect's merge and macro features allow many document variations to be produced from a single template. The following is a fragment of an adaptable requirements component represented using WordPerfect:

```
~a. {VARIABLE}ssid~ configuration. The {VARIABLE}appl~application soft-
ware and the interprocessor communications software in the {VARIABLE}ssid~
subsystem will be stored in ROM and executed on an {COMMENT}
~{IF}{FIELD}ssproc~=1~Intel80186 {ENDIF}{COMMENT}
~{IF}{FIELD}ssproc~=1~Intel80286 {ENDIF}{COMMENT}
~{IF}{FIELD}ssproc~=1~Intel80386 {ENDIF}{COMMENT}
processor. The {VARIABLE}ssid~ subsystem contains {VARIABLE}ssrom~K of
ROM and {VARIABLE}ssram~K of RAM. The {VARIABLE}appl~ application
uses the MIL-STD-1553B bus to exchange messages with applications existing in other
subsystems in the network. The {VARIABLE}appl~ application communicates over
the MIL-STD-1553B bus(es) via the terminal(s) described below.
```

• *Adaptable Design Components.* The design components are used to produce software design specifications for systems in the subdomain. The content of these components was determined from the standardized design developed during the subdomain analysis. As with the adaptable requirements components, customer requirements determined the form of the work products and we implemented these components using WordPerfect.

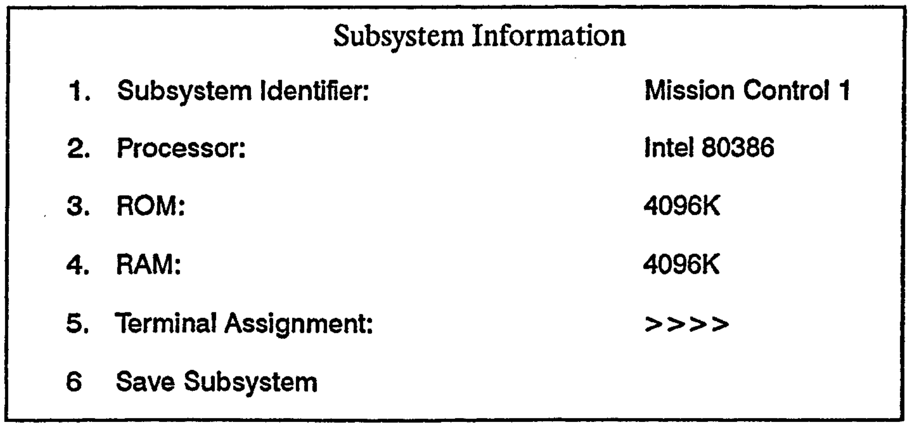
• *Adaptable Code Components.* The code components were developed in Ada. We used Ada generics and the Consortium's TRF2 metaprogramming notation as mechanisms to represent the variation in these components. Due to resource limitations, we only implemented a subset of the components that were called for in the standardized design. We implemented a sufficient subset to allow us to produce pieces of two systems that varied significantly and were similar to others built previously.

• *Automated Support for the Application Engineering Process.* The automated support we produced consists of an environment that implements part of the standardized development process defined during the subdomain analysis. The environment consists of a graphical interface and a generation facility, both developed using the WordPerfect merge and macro functions. The graphical interface, which consists of a sequence of menus and screens, allows the engineer to create, modify, and browse through an application model. The specification language is based directly on the decision model for the subdomain. The generation facility uses the application model to guide the automatic generation of a software requirements document from adaptable requirements components. The adapted documentation is generated by performing a WordPerfect merge of the adaptable requirements components and the engineer's decisions from the application model. Our intention is to integrate access to the adaptable design and code components into this environment so that an application engineer can use the environment to automatically produce requirements, design, and implementations for systems in the domain. One of the screens from the graphical interface, corresponding to the decision class shown in Figure 2, is shown in Figure 3. The following shows a portion of a generated software requirements specification that was produced by tailoring the previously shown adaptable component using the decision choices shown in Figure 3:

> 1.2.1.1. Mission Control 1 configuration. The RICC application software and the interprocessor communications software in the Mission Control 1 subsystem will be stored in ROM and executed on an Intel 80386 processor. The Mission Control 1 subsystem contains 4096K of ROM and 4096K of RAM. The RICC application uses the MIL-STD-1553B bus to exchange messages with applications existing in other subsystems in the network. The RICC application communicates over the MIL-STD-1553B bus(es) via the terminal(s) described below.

The domain implementation supports an improved process for specifying and producing application systems in the MIL-STD-1553B communication software domain. Engineers use the graphical interface to specify systems in terms of high-level requirements and engineering decisions that distinguish them from other systems in the domain. When the engineer is confident that the application model is correct, the environment allows him to generate software requirements directly from his application model. The environment generates the software requirements documentation by tailoring reusable requirements components with the engineer's decisions as captured in the application model. The engineer generates design documentation and the implementation by manually selecting, adapting, and composing adaptable code components. This process is guided by the standardized design and the application model and is completely mechanical. The application engineering process is iterative: if after reviewing the generated products, the engineer realizes that the described system is not exactly what was desired, he can quickly go back and modify the specification to suit his needs and then regenerate the product.

Validation. To validate our domain implementation, we used our application engineering environment to specify and produce the requirements for the MIL-STD-1553B communication software for two CCM systems similar to ones previously developed by Rockwell. These systems differed significantly from each other, yet we had no problems specifying them and generating corresponding


Subsystem Information

1.	Subsystem Identifier:	Mission Control 1
2.	Processor:	Intel 80386
3.	ROM:	4096K
4.	RAM:	4096K
5.	Terminal Assignment:	>>>>
6	Save Subsystem	

Figure 3. A Screen From the Application Engineering Environment

software requirements documentation. This convinced us of the feasibility of automated support for the specification of systems in terms of requirements decisions and for the generation of software work products from adaptable components.

The completed application engineering environment (requirements generation only) was widely demonstrated to Rockwell engineers and management. It was also used as a part of a formal inspection conducted by domain experts (engineers with substantial experience developing software for the 1553B bus) on the subdomain work products. This exposure resulted in many refinements to the domain analysis work products.

Project Use. Our domain implementation has been used on a CCSD project for a communication and signal processing application. A number of circumstances affected the way in which the domain implementation was used. First, the decision to use the domain implementation was made after the project was in the design phase; therefore, the opportunity to generate requirements documentation had passed. Second, the project proceeded concurrently with our development of the adaptable design documentation components so that these were not available for project use. As a result, the application engineering process followed was more conventional in nature than what we expect in the future.

The application project described its target system using our specification notation and then used this to instantiate our standardized design for the target system. On the whole, the standardized design served the project well, and only minor modifications due to missing variations were required. These variations are to be incorporated into the domain work products in support of future systems. Additionally, the project was able to reuse some of the adaptable code components in the domain implementation. A higher level of code reuse could have been obtained if the target system had included more of the implemented portions of the standardized design. However, because the project implemented missing code according to the standardized design, it can be folded into the domain to expand the domain implementation for use on future projects. Even though this use of the domain implementation was less than ideal, it still resulted in significant savings through schedule compression.

The availability of the 1553B domain implementation also influenced the bid and proposal process for one of CCSD's new programs. The primary influences were to the technical approach and to cost projections. The technical approach was affected in that certain technical decisions were made because they corresponded to variations supported in the domain and, therefore, could be supplied at lower cost. The dollar amount of the bid was reduced because the domain implementation gives CCSD the capability to produce the software and deliverables for the target system at a lower cost.

Lessons Learned

The future of the software-intensive systems marketplace belongs to those organizations that can profitably deliver high-quality products in response to diverse or rapidly changing customer needs. The ability to do that comes fundamentally from having people with the expertise and knowledge to understand those needs and create effective solutions. A Synthesis process is one part of a developing system or software engineering discipline that can help an organization better leverage its employees' efforts to achieve success. There are several lessons from our pilot experience with Synthesis:

- *A Synthesis approach has immediate benefits.* Although the original conception of Synthesis was motivated by a vision of fundamental improvements in application development based on domain-specific reuse, there are benefits that can accrue from the beginning of a transition to Synthesis practices. A first potential benefit is better support for bid and proposal efforts based on a clearer, shared understanding of the organization's existing capabilities to deliver a particular type of system and a derivative ability to more accurately distinguish and estimate parts of a system that offer (lower cost) reuse opportunities from parts that will require (higher cost) new development. A second potential benefit is the existence of an explicit standard definition of requirements, arising from a coherent agreement on business concepts and terminology, that can be used as a starting point in identifying and resolving unclear or incomplete requirements statements from customers. A third potential benefit is the institutionalization of shared knowledge and expertise about systems in a business area that can more rapidly raise the working level of new or less experienced personnel. A fourth potential benefit is that there may already be an opportunity to standardize and use many existing software, test, or documentation components because there is often an existing but unrecognized basis for a standardized design. Each of these is achievable without adoption of a complete Synthesis approach.

- *A management commitment at all levels is essential.* A domain is only as good as the people assigned to develop and maintain it. These people must be knowledgeable and experienced in all aspects of customer problems and their solutions. If their immediate and middle managers do not see the value in the effort, these managers will take the first opportunity to move key people to other assignments. If upper managers do not see the value in the effort, there will not be sufficient funding to do a proper job. Additionally, the involvement of respected technical and management leaders is critical to gaining the confidence of project managers who must use a domain for any investment to pay off.

- *Making reuse pay requires a substantial commitment of effort.* Some organizations would like to take the view that reuse should entail, at most, a small up-front effort and instructions to coders on 'how to reuse'. The truth is that a domain can provide significant leverage for an organization's projects but only in proportion to the effort invested in developing and maintaining it. A reasonable starting point is to realize that creating a domain, whatever its scope in terms of system size and complexity, cannot possibly cost less (or necessarily much more) than it would to handcraft yet another system within that scope; incremental commitment of course can spread costs and risks over time. The payoff comes not from minimizing investment but from an acquired ability to rapidly deliver new or changed systems.

- *Adoption of a disciplined engineering process stimulates adoption of disciplined engineering methods.* When CCSD managers and engineers recognized the value of Synthesis as a disciplined engineering process, they discovered that attaining its full potential required acceptance of disciplined engineering methods as well. Synthesis does not dictate particular methods and, in fact, only requires (as a minimum) standardization of the way products are represented. However, Synthesis provides a framework within which organizations can adopt disciplined methods without necessarily having to retrain every engineer.

Conclusions

Rockwell has used Synthesis successfully to create a domain for CCM MIL-STD-1553B system bus communication software. We demonstrated that a knowledgeable engineer can describe systems in this domain in terms of critical requirements and engineering decisions and that those specifications can guide the generation of corresponding software work products from adaptable components. The validation exercises and limited project use of the domain implementation convinced us that the practice of Synthesis in well-chosen business areas can result in significant improvements in how CCSD builds software. We believe that a process for domain-specific specification of a system and the generation of a corresponding product will increase CCSD's productivity and product quality for systems in supported domains.

The pilot project developed an environment that could satisfy needs of existing projects. The domain implementation has been used successfully on a current project, resulting in improved productivity, and has also permitted CCSD to be more competitive on a bid for a new project. Further subdomain work is ongoing. We hope, in the future, to integrate access to the adaptable design and code components into the application engineering environment. This will enable an application engineer to use the environment to automatically produce requirements, design, and implementations for systems in the subdomain. We also expect that, as the environment is used on more projects, there will be significant feedback that will enable further refinement of the domain implementation.

Based on the success of this pilot project, Rockwell CCSD has committed to investigate Synthesis for production use by its Multimedia Message Handling Systems (M^3HS) development group. Developing a standardized product line and supporting process for this business area will facilitate large-scale reuse and will allow this development group to be more competitive and profitable. This group has predicted sales of as many as 20 M^3HS systems plus upgrades in the near future, and the group believes that Synthesis is critical to being able to meet this demand.

Acknowledgments

The results presented in this report would not have been possible without the contributions of the other participants in this project. Special thanks go to John Allen, Cheryl Chin, Phil Garcia, and Doug Johnson of Rockwell International. Rich McCabe of the Software Productivity Consortium was instrumental in helping Rockwell initiate the pilot project and, along with Wil Spencer, Roger Williams, and Steve Wartik of the Consortium and Joe Keffer of Rockwell, provided comments that significantly improved this report.

References

1. G. Campbell, S. Faulk, and D. Weiss, *Introduction to Synthesis*, INTRO_SYNTHESIS_PROCESS-90019-N, Software Productivity Consortium, Herndon, Va., 1990.
2. E. Dijkstra, "Notes on Structured Programming," *Structured Programming*, Academic Press, London, 1972, pp. 1-82.
3. D. Parnas, "On the Design and Development of Program Families," *IEEE Trans. Software Eng.*, March 1976, pp. 1-9.
4. Software Productivity Consortium, *Reuse-Driven Software Processes Guidebook*, SPC-92019-CMC, Software Productivity Consortium, Herndon, Va., 1993.
5. C. Cleaveland, "Building Application Generators," *IEEE Software*, July 1988, pp. 25-33.
6. Software Productivity Consortium, *TRF2 Metaprogramming Tool User Guide*, SPC-91132-MC. Software Productivity Consortium, Herndon, Va., 1991.
7. D. Parnas, P. Clements, and D. Weiss, "The Modular Structure of Complex Systems," *IEEE Trans. Software Eng.*, March 1985, pp. 259-266.
8. U.S. Department of Defense, *Digital Time Division Command/Response Multiplex Data Bus*, MIL-STD-1553B, U.S. Department of Defense, Washington, D.C., 1978.
9. Software Productivity Consortium, *ADARTS Guidebook*, SPC-91104-MC, Software Productivity Consortium, Herndon, Va., 1991.